Sphere Pack Documentation

INTRODUCTION

This module provides access through Python to the collection of Fortran programs in SPHEREPACK 3.0, which is a collection of programs produced at the National Center for Atmospheric Research by John C. Adams and Paul N. Swarztrauber for computing certain common differential operators and performing related manipulations on a sphere. It provides solutions via the spectral method that uses both scalar and vector harmonic transforms. Since scalar and vector quantities are fundamentially different on the sphere due to the multiple valued and discontinuous nature of vectors at the poles, separate functions are provided for scalar and vector quantities.

RESTRICTIONS

Spherepack is not for everyone. Spherepack manipulates data defined strictly on the global sphere. Accordingly, the following restrictions apply:

The longitude vector consists of unique evenly spaced points spanning the globe.
The latitude vector can be either gaussian or evenly spaced including the poles.
Missing data is not allowed.

THE SHORT STORY

To save time in coming up to speed in the use of this python wrapper around SPHEREPACK 3.0, skip to section GENERAL EXAMPLE. To run some test cases using analytically generated data type
python sphere.py

It also puts out this documentation in the file spheremodule.doc and a copy of the information written to the screen as screen.asc for what it is worth.

ORGANIZATION

This module is object oriented for simplicity. It is organized as three classes reflecting the three functional groups in SPHEREPACK 3.0 which perform vector analysis computations, regridding and grid shifting. The vector analysis computations are contained in the Sphere class, the regridding in the Regrid class and the grid shifting in the Shiftgrid class. The class names begin with a capital letter. Python is case sensitive. Access to the functions housed in the classes is a simple two step process. The first step is making an instance of the class which hosts the desired functionality. The second step is calling the actual function or functions of interest.

Making an instance requires passing the grid vectors and possibly a computaional parameter selecting the storage treatment of the Legendre polynomials. This information is used internally to select the Fortran function call. For example, to compute the divergence of a vector function, it is necessary to choose one of four Fortran functions according to whether the grid is evenly spaced or gaussian and the Legendre polynomials are stored or computed. These bookkeeping details are done automatically for the user based on the argument list passed in the instance statement.

Having set up the machinery with the instance call, the class method functions are called by qualification using the dot operator. It is only necessary to submit the data and accept the return of the answer.

As a preview, here is the two step process for calculating the divergence and the vorticity of a
2D vector field (u, v)

```
x = sphere.Sphere(longitudeVector, latitudeVector)
divergence = x.div(u, v)
vorticity = x.vrt(u, v)
```

which demonstrates the advantage of the object oriented approach. The instance can be reused for
multiple computations and the argument list is simple and intuitive. Having calculated the
divergence and the vorticity, the user may want do view smoother fields. To truncate the fields
at T16, write

```
divergence_T16 = x.truncation(16, divergence)
vorticity_T16 = x.truncation(16, vorticity)
```

DOCUMENTATION

Documentation written to the file spheremodule.doc can be obtained after importing the spheretest module by typing

```
spheretest.document()
```

A brief view of the documentation consisting of the overview can be written to the file spheremodule.doc after importing the sphere module by typing

```
spheretest.document(brief = 'yes')
```

Online documentation for individual classes or method functions is available from the module doctring and the
help package. As examples:

```
print sphere.__doc__ -- four page overview of the package
sphere.help() -- contents of the help function
sphere.help('div') -- documentation for the Sphere class div method
sphere.Sphere.div.__doc__ -- documentation for the Sphere class div method
```

For the utilities type for example

```
sphere.help('gridGenerator') -- grid vector generation
```

CONTENTS

This module provides access to the Fortran library in terms of three functional groups
which perform vector analysis computations, regridding and grid shifting.

Vector Analysis and Truncation -- contained in Sphere class

The functions for computing differential operations and their inverses on scalar
and vector functions in spherical coordinates on a global grid are:

div -- computes the divergence of a vector function

idiv –– inverts the divergence creating an irrotational vector function
vrt –– the vorticity of a vector function
ivrt –– inverts the vorticity creating a divergence_free vector function
idvt –– inverts the divergence and the vorticity creating a vector function

vts –– computes the derivative of the vector function with respect to latitude
grad –– computes the gradient of a scalar function
igrad –– inverts the gradient creating a scalar function
slap –– computes the Laplacian of a scalar function
islap –– inverts the Laplacian of a scalar function
vlap –– computes the Laplacian of a vector function
ivlap –– inverts the Laplacian of a vector function
sfvp –– computes the stream function and the velocity potential of a vector function
isfvp –– inverts the stream function and the velocity potential of a vector function

One additional function, not part of the basic library, has been added to perform triangular truncation with or withhout tapering:

truncation–– truncates scalar or vector data at specified total wavenumber

The basic functions for spectral analysis and synthesis directly accessible from python are:

sha –– computes the spherical harmonic analysis of a scalar function
shs –– computes the spherical harmonic synthesis of a scalar function
vha –– computes the spherical harmonic analysis of a vector function
vhs –– computes the spherical harmonic synthesis of a vector function

Regridding –– contained in Regrid class

The two functions for regridding are:

regridScalar –– transfers scalar data from one global grid to another
regridVector –– transfers vector data from one global grid to another

Shifting –– contained in Shiftgrid class

The two functions for shifting an evenly spaced grid by half an increment in longitude and latitude are:

shiftScalar –– transfers scalar data between an evenly spaced regular and an offset grid
shiftVector –– transfers vector data between an evenly spaced regular and an offset grid

where the regular grid is defined as one which includes the poles.

Utilities not part of the overall scheme but still of possible interest

gridGenerator –– generates the longitude and latitude vectors
truncate –– provides truncation at the spectral coefficient level

PROCEDURE

Access to the Spherepack Fortran library is provided through the module spherepackmodule.so which has been constructed using the Pyfort utility. A wrapper around the functions in this spherepackmodule.so has been created as sphere.py. Assuming path access to spherepackmodule.so and sphere.py, to use Spherpack3.0 from the National Center for Atmospheric Research, it is only necessary to import the sphere.py module. It contains the classes Sphere, Regrid, and Shiftgrid designed for the user. This is done by typing at the python prompt

```
import sphere
```

Use of spherepack is a two step process. The first step is the creation of an instance of the appropriate class for the computation at hand. For this step there are three classes as a choice: Sphere, Regrid and Shiftgrid. The second step is the selection of the class method of interest by qualification with the dot operator.

As notation related to the display of the argument lists in the calls, the required ones use the keyword name only and the optional ones are written as keyword entries.

In practise there are lots of choices allowed by Python. At one extreme all arguments can be submitted using keyword assignments while setting them to None for optional unused vectors. In the class Sphere, the computational scheme default 'computed' can be changed to 'stored'. At the other extreme all arguments can be submitted in the correct position. The examples will clarify this.

Vector Analysis and Truncation

As the first step for the spherical differential operation calculations, the instance x of the Sphere class is made with the statement typed to the python prompt

```
x = sphere.Sphere(lonArray , latArray, numberLevels = nlev, numberTimes = ntime,

computed_stored = 'computed')
```

where nlev and ntime are the actual number of levels and times respectively and the keywords are

lonArray = longitude vector (required)
latArray = latitude vector (required)
numberLevels = number of levels (optional)
numberTimes = number of times (optional)
computed_stored (optional) : 'computed' –– computed Legendre polynomials
'stored' –– stored Legendre polynomials
This choice involves a 30% storage/speed tradeoff

The instance request uses computed_stored and the grid vectors to associate the actual Fortran calls with the requested calculation. This association is automatic and need not concern the user. If the longitude and latitude grid vectors are not available, prior to the instance creation they can be constructed by calling the utility function girdGenerator.

As a second step, the desired method is selected. As a concrete example, the divergence of the the vector pair u and v is found with

```
div = x.div(u, v)
```

or
div = x.div(u, v, missingValue) – with the request to check for missing data

This div function calls all the spherepack functions needed to return the divergence. The second form looks for the presence of missing data and returns an error if found. It uses the exact value so it must be the value in the data file.

Regridding

As the first step for regridding make an instance x of the Regrid class with

x = sphere.Regrid(lonArrayOut, latArrayOut, lonArrayIn, latArrayIn, numberLevels = nlev, numberTimes = ntime)

where nlev and ntime are the actual number of levels and times respectively and the keywords are

lonArrayOut = output grid longitude vector (required)
latArrayOut = output grid latitude vector (required)
lonArrayIn = input grid longitude vector (required)
latArrayIn = input grid latitude vector (required)
numberLevels = input grid number of levels (optional)
numberTimes = input grid number of times (optional)

As an example of the second step, regridding a scalar function sf is accomplished with

sf = x.regridScalar(sf)
or
sf = x.regridScalar(sf, missingValue)

The second form looks for the presence of missing data and returns an error if found. It uses the exact value so it must be the value in the data file.

Shifting

For the grid shifting as step one make an instance x of the Shiftgrid class with

x = sphere.Shiftgrid(lonArray, latArray, numberLevels = nlev, numberTimes = ntime)

where nlev and ntime are the actual number of levels and times respectively and the keywords are

lonArray = longitude vector (required)
latArray = latitude vector (required)
numberLevels = number of levels (optional)
numberTimes = number of times (optional)

As an example of the second step, shifting a scalar function sf is accomplished with

sf = x.shiftScalar(sf)
or
sf = x.shiftScalar(sf, missingValue)

The second form looks for the presence of missing data and returns an error if found. It uses the exact value so it must be the value in the data file.

GENERAL EXAMPLE

Step 1. Type
import sphere

Step 2. From this documentation determine the class which offers the desired computation. There are only three choices: the Sphere class, Regrid class and Shiftgrid class to use in ClassName below.
Instructions on making an instance is obtained by typing

```
sphere.help('ClassName')
```

Step 3. Make an instance, x, of the specific class ClassName using the statement

```
x = sphere.ClassName(argument1, argument2, .........)
```

Step 4. Perform the actual computation using a specific function named functionName, which has been identified from this documentation.

```
returned values = x.functionName(argument1, argument2, .........)
```

To get information about the agrument list type

```
sphere.help('functionName')
```

TESTING

Typing

```
cdat spheretest.py
```

generates some testing of the spheremodule using analytical functions as fields.

For additional testing using real geophysical data, you might try the following exercise.

Step 1. Get winds u and v and their grid vectors, longitude values (lonvals) and latitude values (latvals), from somewhere.This example uses 2D fields for simplicity. The fields must be global without missing values.

Step 2. Make an instance of the Sphere class, x, as

```
x = sphere.Sphere(lonvals, latvals)
```

Step 3. Compute the streamfunction, sf, and the velocity potential, vp, using

```
sf, vp = x.sfvp(u, v)
```

Step 4. Compute the source for the streamfunction, sf_source, and the velocity potential, vp_source, using the scalar Laplacian

```
sf_source = x.slap(sf)
vp_source = x.slap(vp)
```

Step 5. Compute the source for the streamfunction, vort, and the velocity potential, div, directly using the divergence and the vorticity

```
vort = x.vrt(u, v)
div = x.div(u, v)
```

Step 6. Compare the results for equality, sf_source with vort and vp_source with div. If the comparison fails, please complain about it.